HAAGA-HELIA
ammattikorkeakoulu

PDF-dokumenttien tuottaminen .NET-ympäristössä

Tommi Piironen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2013

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

17.1.2013

Tietojenkäsittelyn koulutusohjelma

Tekijä Tommi Piironen	**Aloitusvuosi** 2009
Raportin nimi PDF-dokumenttien tuottaminen .NET-ympäristössä	**Sivu- ja liitesivumäärä** 27
Ohjaaja Jukka Juslin	

Tämän opinnäytetyön tarkoituksena on tuottaa sovellus PDF-muotoisten rahtiliikennedokumenttien tuottamiseen. Sovelluksella tuotetaan lähetysten rahtikirjoja sekä kolli – eli osoitelappuja. Projekti tehdään eräälle Suomessa toimivalle logistiikka-alan yritykselle. Sovellus ohjelmoidaan C#-ohjelmointikielellä käyttäen Microsoftin .NET-ohjelmointiympäristöä.

Sovelluksen tuottamien dokumenttien ulkoasu määritellään XML-tiedostoja käyttäen. Näin luotavien dokumenttien ulkoasua voidaan muokata helposti, joten sovelluksen ylläpidettävyys helpottuu.

Opinnäytetyössä esitellään myös .NET-ympäristön rakenne sekä kuvaillaan C#-ohjelmointikielen toimintaa.

Projektin aikana valmistuneella sovelluksella pystyy tuottamaan lähetysten rahtikirjat sekä A5-kokoisia kollilappuja. Sovelluksen käyttöönotto ja integrointi tilaajan tietojärjestelmään eivät kuuluneet tämän projektin piiriin.

Asiasanat
C#, .NET, XML, ohjelmistokehitys

HAAGA-HELIA
ammattikorkeakoulu

Abstract

17.1.2013

Degree Programme in Business Information Technology

Authors Tommi Piironen	**Year of entry** 2009
The title of thesis Generating PDF-documents with .NET-framework	**Number of pages and appendices** 27
Supervisor(s) Jukka Juslin	

The objective of this thesis is to produce a software component for producing pdf-documents. The component will produce waybills and package labels used in freight transportation. The software is programmed using Microsoft's .NET-framework with C#-language.

The layouts of the pdf-documents are defined by using XML-files, so modifying or even adding new layouts is simple and thus the maintainability of the software is easier.

This thesis will also give a brief tour of the .NET-framework and C#-programming language.

The software finished during the project can create waybills and A5-sized package labels. The product's installation and integration to the existing systems was not part of this project.

Key words
C#, NET, XML

Sisällys

1 Johdanto 1

2 Vaatimusmääritys 1

2.1 Taustaa 1

2.2 Käyttötapaukset 3

2.2.1 Rahtikirjan tuottaminen 3

2.2.2 Osoitelapun tuottaminen 4

2.2.3 Asiakaskohtaisen osoitelapun tuottaminen 4

3 Teoriatausta 4

3.1 .NET-Framework 4

3.1.1 .NET-ympäristön toiminta ja rakenne 6

3.1.2 C# 7

3.1.3 Visual Studio 8

3.2 XML 9

4 Järjestelmän toteutus 10

4.1 XML-asetustiedostojen käyttö 10

4.2 PDF-dokumenttien tuottaminen 12

4.3 Sovelluksen toiminta ja rakenne 13

4.3.1 Lähetystietojen hakeminen tietokannasta 14

4.3.2 Pdf-komponentin rakenne 15

4.3.3 PDFManager 17

4.3.4 ConsignmentData 17

4.3.5 BaseXmlParser 18

4.3.6 XmlElement 18

4.3.7 Settings 19

4.3.8 SettingsManager 19

4.3.9 DocumentWriter 20

4.3.10 DataBaseAccess 21

4.4 Tietokanta 21

4.5 Sovelluksen testaaminen 22

5 Pohdinta 23

5.1 Projektin tulokset 23

5.2 Sovelluksen jatkokehitysehdotuksia 25
5.3 Projektin aikana opittua 25
Lähteet 27

1 Johdanto

Tämän opinnäytetyön tarkoituksena on luoda ohjelmistokomponentti pdf-muotoisten dokumenttien tuottamiseen. Komponentti ohjelmoidaan käyttäen Microsoftin .NET-ohjelmointiympäristöä C#-ohjelmointikielellä käyttäen .NET 4.0-versiota. Sovelluskehitysympäristönä toimii Visual Studio 2010.
Opinnäytetyö tehdään eräälle Suomessa toimivalle logistiikka-alan yritykselle. Yritys on uudistamassa asiakkailleen tarjoamiaan verkkopalveluitaan ja tämän projektin puitteissa luotava ohjelmistokirjasto on osa tätä uudistamishanketta. Verkkopalvelun kautta asiakas voi mm. tehdä kuljetustilauksia sekä seurata niiden toimituksen etenemistä. Kun asiakas on tehnyt kuljetustilauksen, hänen on mahdollista ladata lähetyksen rahtikirja sekä kolli – eli osoitelaput pdf-muodossa, jotka hän voi tulostaa itselleen. Ohjelmistokomponentilla tuotetaan asiakkaille nämä dokumentit.
Työn tilaajalla on olemassa tällä hetkellä sovellus pdf-dokumenttien luontiin, mutta se on toteutettu vanhalla. NET-versiolla, eikä sen tuottamien dokumenttien ulkoasua voi muokata eikä uusia dokumenttityyppejä lisätä muokkaamatta ja kääntämättä sovelluksen ohjelmakoodia. Tilaaja toivoo, että uudessa versiossa olisi mahdollista muokata luotavien dokumenttien ulkoasua helposti sekä lisätä aivan uusia dokumenttivaihtoehtoja muokkaamatta ohjelmakoodia, jolloin sovelluksen ylläpidettävyys helpottuisi.

2 Vaatimusmääritys

2.1 Taustaa

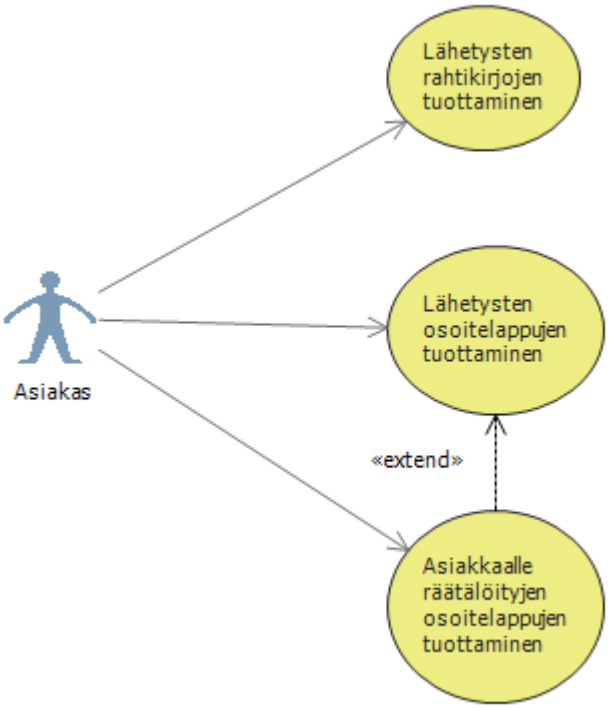
Ohjelmistokomponentti tuottaa asiakkaan tekemille kuljetustilauksille niiden rahtikirjat sekä osoite -eli kollilaput pdf-muotoisina dokumentteina.
Ohjelmistokomponentti on osa yrityksen verkkopalveluiden uudistamishanketta. Se ei ole itsenäinen ohjelmisto, vaan eräänlainen ohjelmistokirjasto, jota verkkosovellus kutsuu, kun asiakas on tehnyt kuljetustilauksen. Tämän jälkeen komponentti luo asiakkaalle hänen pyytämänsä dokumentin ja palauttaa sen verkkosovellukselle, joka puolestaan tarjoaa sen asiakkaalle ladattavaksi. Komponentin lopullisessa versiossa sitä kutsutaan web service-rajapinnan yli. Sovelluksella ei siis ole minkäänlaista graafista käyttöliitty-

mää. Web servicen toteutus ei kuitenkaan kuulu tähän projektiin vaan se toteutetaan myöhemmin.

Koska tämä projekti tulee valmistumaan ennen kuin uusi verkkopalvelu saadaan käyttöön, tullaan projektin aikana valmistuva sovellus ottamaan käyttöön myös nykyisessä verkkopalvelussa. Niinpä sovellus tulee suunnitella siten, että se on yhteensopiva sekä uuden että nykyisen verkkopalvelun kanssa.

Kun verkkosovellus kutsuu komponenttia, sille välitetään samalla käytettävän lähetyksen tiedot, joiden perusteella pdf-dokumentti luodaan. Koska uuden verkkopalveluiden kehitys on vielä kesken, ei tietokannan käyttämiseen tarkoitettua rajapintaa ole vielä kehitetty, joten tämän opinnäytetyön aikana valmistuva sovellus hakee lähetyksen tiedot itse suoraan tietokannasta. Sovelluksen tulee pystyä hakemaan tiedot sekä nykyisen verkkopalvelun tietokannasta että myös työn alla olevan uuden palvelun tietokannasta. Luodut dokumentit tallennetaan tietokantaan, jottei niitä tarvitse luoda uudestaan toistamiseen.

2.2 Käyttötapaukset



Kuva 1. Käyttötapauskaavio

Kuva 1 esittelee ohjelmistokomponentin käyttötapaukset. Ohjelmistokomponentin tarkoituksena on tuottaa asiakkaalle lähetyksen rahtikirjan sekä osoitelaput.

2.2.1 Rahtikirjan tuottaminen

Rahtikirja on rahtiliikenteessä käytettävä dokumentti, joka kuvaa lähetyksen osapuolet sekä itse rahdin sisällön. Lähetyksen osapuoliin kuuluvat lähetyksen lähettäjä, vastaanottaja sekä lähetyksen nouto- ja toimitusosoite. Rahtikirja kuvaa myös itse lähetyksen sisällön eli lähetyksen painon ja kollien, eli ”pakettien” tai muiden kuljetusyksiköiden, lukumäärän sekä mittasuhteet. Näitä kollikohtaisia tietoja kutsutaan kolliriveiksi. Suomessa yleisesti käytetty rahtikirjastandardi on SFS 5865, johon pohjautuvaa rahtikirjamallia myös projektin tilaaja käyttää. Ohjelmistokomponentin tuottamat rahtikirjat pohjautuvat siis myös tähän standardiin. Kappaleessa 5.1 esitellään projektin aikana luodulla komponentilla tuotettu standardin pohjautuva rahtikirja. Rahtikirjat ovat aina A4-kokoisia. Rahtikirjoja tuotetaan aina vähintään kolme kappaletta, yksi kappale lähet-

täjälle, vastaanottajalle ja rahdin kuljettajalle. Tarvittaessa rahtikirjasta voidaan tuottaa useampia kopioita muille osapuolille.

2.2.2 Kollilappujen tuottaminen

Osoite – eli kollilappu on lähetyksen kolleihin kiinnitettävä lappu, joka sisältää lähetyksen lähettäjän tiedot sekä toimitusosoitteen. Ohjelmistokomponentin tuottamat kollilaput ovat GS1-standardiin pohjautuvia, joka on yleinen Suomessa käytetty standardi. Standardi määrittää vain, mitä tietoja lapussa on oltava, joten osoitelapun koko tai muoto voi siis vaihdella. Tuotettavat osoitelaput voi myös olla erikokoisia. Kokovaihtoehtoina ovat esimerkiksi A4 – koko ja sekä useat muut pienemmät koot. Osoitelappuja tuotetaan aina yhtä monta kuin lähetyksessä on kolleja

2.2.3 Asiakaskohtaisen osoitelapun tuottaminen

Asiakkaat saattavat haluta heille erikseen muotoiltuja kollilappuja. Koska osoitelappustandardi ei määritä lapun tarkkaa ulkoasua, tämä on mahdollista toteuttaa. Asiakaskohtaiset laput voivat poiketa normaalista kokonsa tai dokumentin ulkoasun suhteen. Kaikista osoitelapuista on kuitenkin löydyttävä standardin edellyttämät osoitetiedot.

3 Teoriatausta

Tässä kappaleessa esitellään projektissa käytetyt teknologiat, joista tärkeimmät ovat .NET-Framework, C#-ohjelmointikieli sekä XML-merkintäkieli. Kappaleessa esitellään myös .NET-ohjelmoinnissa käytettävä Visual Studio-ohjelmointiympäristö.

3.1 .NET Framework

.NET Framework on Microsoftin kehittämä ohjelmointiympäristö Windows-käyttöjärjestelmille (Nagel, Evjen, Glynn, Watson, Skinner 2010, LII). .NET tukee ohjelmointia useilla eri ohjelmointikielillä, joista yleisimmät ovat C#, Visual Basic ja Visual C++. Näistä ohjelmointikielistä C# on nimenomaan kehitetty .NET-ympäristöä ajatellen, Visual Basic ja Visual C++ ovat .NET:ää vanhempia ohjelmointikieliä, jotka on

kuitenkin muokattu .NET-yhteensopiviksi.(Nagel ym. 2010, 5.) Myös useat muut tahot ovat kehittäneet eri ohjelmointikielistään .NET-yhteensopivia versioita.

Ensimmäinen versio .NET-ympäristöstä julkaistiin vuonna 2002. Microsoft on sen jälkeen jatkanut .NET:n kehitystä aktiivisesti ja lisännyt uusia ominaisuuksia ja ohjelmistokirjastoja. Tällä hetkellä viimeisin versio .NET:stä on versio 4.5, joka julkaistiin vuoden 2012 syyskuussa. Eri versiot ovat taaksepäin yhteensopivia, eli uudemmalla .NET-versiolla voi ajaa vanhemmalle .NET-versiolle ohjelmoituja sovelluksia. (MSDN a.)

.NET sisältää ohjelmakomponentteja mm. verkko -ja työpöytäsovellusten sekä web service-palveluiden tekemiseen. Verkkosovellusten ohjelmointiin käytetään ASP.NET-kirjastoa. ASP.NET-sovelluksissa yksitäiset verkkosivut on linkitetty omaan ohjelmistoluokkaansa, joka sisältää sivuston toiminnallisuuden. Tämä sovellusluokka generoi html-sivun sisällön dynaamisesti. Kun sivulta esimerkiksi lähetetään lomake, lomakkeen datan käsittely toteutetaan tämän luokan sisällä. (Nagel ym. 2010, 17-18.) ASP.NET on ollut mukana .NET-ympäristössä jo ensimmäisestä versiosta alkaen ja se oli pitkään käytännössä ainoa keino tehdä verkkosovelluksia .NET ympäristössä. ASP.NET-tekniikassa on kuitenkin muutamia ongelmia, joista ehkä suurin on se, että HTML-sivu ja sen toiminnallisuudet sisältävä sovellusluokka ovat tiukasti yhteydessä toisiinsa. Tämän vuoksi sovelluksen testaaminen esimerkiksi yksikkötestein on hankalaa. Tämä oli yksi syy sille, että Microsoft kehitti ASP.NET MVC-ohjelmistokirjaston, joka erottaa sovelluksen toiminnallisuuden MVC-mallin mukaisesti mallin, näkymän ja kontrollerin kesken. Näin sovelluksen testaaminen helpottuu. ASP.NET MVC on rakennettu ASP.NET-kirjaston päälle, joten yhteensopivuus vanhempaan teknologiaan on taattu. (Nagel ym. 2010, 1260.)

Työpöytäsovellusten ohjelmoimiseen .NET-ympäristössä käytetään joko Windows Forms tai Windows Presentation Foundation (WPF)-sovelluskirjastoa (Nagel ym, 2010. 983, 1117). WPF on näistä kahdesta uudempi tekniikka, jossa sovellusten käyttöliittymät määritellään XML-syntaksiin perustuvalla XAML-merkintäkielellä. Niinpä sovelluksen käyttöliittymä ja ohjemallinen toiminnallisuus on helppo pitää erillään toisistaan (Nagel ym, 2010, 983). WPF:llä on helpompi tehdä graafisesti vaativia sovelluksia kuin

Windows Forms:lla. WPF-sovelluksiin on helpompi upottaa multimediaa, kuten videoita tai näyttävää grafiikkaa (MSDN 2006.) Esimerkiksi uusimmat versiot Microsoftin kehittämästä .NET-ohjelmistokehityssovelluksesta Visual Studiosta on ohjelmoitu käyttäen WPF-sovelluskirjastoa (Mackey 2010, 9).

Windows Forms on ollut .NET-ympäristössä mukana jo heti sen ensimmäisestä versiosta lähtien ja se onkin saanut vaikutteita Visual Basic:llä tehdystä työpöytäohjelmistokehityksestä. (Nagel ym. 2010, 1117.)

Web service-palveluiden toteuttamiseen .NET-ympäristössä käytetään Windows Communication Foundation eli WCF-kirjastoa. Aiemmin .NET:ssä oli käytössä useita erillisiä, osin päällekkäisiä ratkaisuja alustariippumattoman verkkoviestinnän toteuttamiseen palvelimen ja asiakkaiden välillä. WCF sisältää kaikkien näiden tekniikoiden ominaisuudet ja tarjoaa uusia toiminnallisuuksia. (Nagel ym. 2010, 1279.) WCF:tä käytetään etupäässä SOAP-muotoiseen verkkoviestintään, jossa palvelun lähettämä verkkoliikenne on XML-muotoista. WCF:llä on kuitenkin myös mahdollistaa toteuttaa RESTful-verkkopalveluita, joissa verkkoliikenne toimii http-protokollan yli ja palvelun lähettämät viestit ovat JSON-muotoisia. (Nagel ym. 2010, 1282–1283.)

3.1.1 .NET-ympäristön toiminta ja rakenne

.NET-ympäristössä sovellusten ajamisesta vastaa Common Language Runtime eli CLR. CLR on ajoympäristö, jossa .NET-sovelluksia ajetaan. (Nagel ym. 2010, 4.)
Sovellusten kääntäminen lähdekoodista konekielelle on .NET-ympäristössä kaksivaiheinen. Ensimmäisessä vaiheessa sovelluksen lähdekoodi käännetään Common Intermediate Language (CIL)-muotoon. CIL-koodi on tavukoodia ja kaikki .NET-yhteensopivat ohjelmointikielet käännetään aluksi CIL-muotoon. Tämän ansiosta kaikki .NET-ohjelmintikielet ovat myös yhteensopivia keskenään. (Nagel ym. 2010, 4.) Esimerkiksi C#-kielellä ohjelmoitu sovellus voi kutsua Visual Basicillä ohjelmoitua luokkia ja päinvastoin (MSDN a).

Toisessa vaiheessa CIL-muotoinen koodi käännetään natiiviksi konekieleksi jonka jälkeen se suoritetaan. CLR on vastuussa tästä käännöksestä sekä myös sovelluksen suorittamisesta. (Nagel ym. 2010, 4).
Microsoftin luomien .NET-kielten lisäksi monet kolmannen osapuolen valmistajat ovat kehittäneet eri ohjelmointikielistä .NET-versioita. Esimerkiksi suositusta Python-kielestä on tehty .NET-ympäristössä ajettava versio nimeltään IronPython (IronPython 2012). Kuten kaikki muutkin .NET-kielet, Python-kieli käännetään aluksi CIL-muotoon, jonka CLR voi sen jälkeen suorittaa.

Koska kaikki .NET-sovellukset käännetään ensin CIL-koodiksi ja vasta sen jälkeen konekielelle, on .NET-ympäristö teoriassa alustariippumaton. Tämän lisäksi Mircosoft on standardoinut .NET-ympäristön, joten järjestelmän spesifikaatio on avoin (ECMA International 2012). CLR voisi teoriassa kääntää CIL-kielen esimerkiksi Linux-yhteensopivaksi konekieleksi. Käytännössä kuitenkin Microsoft tarjoaa .NET-ympäristön vain Microsoft Windows-käyttöjärjestelmille. (Nagel ym. 2010, 4) .NET-ympäristöstä on kuitenkin saatavilla avoimen lähdekoodin toteutuksia, jotka toimivat sekä Windowsilla että eri Unix-pohjaisilla käyttöjärjestelmillä. Näistä tunnetuin on Mono, joka tarjoaa .NET-ajoympäristön mm. Windowsille, Linuxille, Applen iOS-käyttöjärjestelmälle ja Googlen Andrdoid-käyttöjärjestelmälle (Mono 2012). Mono ei kuitenkaan tue kaikkia .NET-ympäristön sovelluskirjastoja. Monosta puuttuu tuki esimerkiksi WPF-kirjastolle, sekä tuki ASP.NET MVC 4-versiolle on tällä hetkellä puutteellinen. (Mono 2012b). Koska Mono ei ole Microsoftin virallisesti tukema, ilmestyvät kaikki .NET:n uudet ominaisuudet Monoon aina myöhemmin kuin Microsoftin viralliseen .NET-ympäristöön.

3.1.2 C#

C# on Microsoftin kehittämä ohjelmointikieli .NET-ympäristöön. Se on saanut vaikutteita C++ -ja Java-ohjelmointikielistä. (Nagel ym. 2010, LV-LVI.) C#:n syntaksi muistuttaa hieman enemmän C++ kuin Javaa. C# on nimestään huolimatta kuitenkin perustoiminnallisuudeltaan lähempänä Javaa kuin C-kieliä, koska sekä Java että C#-sovelluksia ajetaan hallitussa ympäristössä. Java-sovellukset pyörivät Java-

virtuaalikoneessa, ja C#-sovellusten suorituksesta vastaa CLR. (Liberty 2001.) Niinpä toisin kuin esimerkiksi C++:ssa ohjelmoijan ei tarvitse huolehtia muistin varaamisesta tai vapauttamisesta. Tämä auttaa ehkäisemään yleisiä ohjelmointivirheitä, jotka voivat johtavaa muistivuotoihin. Tämän lisäksi hallittu ympäristö lisää sovelluksen tietoturvaa. Toisaalta C#:lla voi kirjoittaa ns. ”ei hallittua koodia” (unmanaged code), jolloin ohjelmoija on itse vastuussa muistin varaamisesta ja vapauttamisesta ohjelman suorituksen aikana. Tämän tyyppisen ohjelmakoodi on tarpeen käytännössä vain erittäin paljon suorituskykyä vaativissa sovelluksissa, tai mikäli sovellusten on käytettävä suoraan Windowsin Win32-rajapintaa. (MSDN 2003.)

C# on olio-ohjelmointikieli, joka on kehitetty nimenomaan .NET-ympäristöä silmällä pitäen. Toisin kuin esimerkiksi Visual Basic tai Visual C++ se ei sisällä vuosien saatossa kertynyttä ylimääräistä, nyt jo vanhanaikaisia ominaisuuksia. **(**Nagel ym. 2010, LV). Kuten mainittua, Java ja C# ovat pinnallisesti katsottuna ominaisuuksiltaan melko samanlaisia, mutta vuosien saatossa C# on lisätty ominaisuuksia, joita ei Javasta löydy. C#:n on esimerkiksi omaksuttu C++:n funktiopointterit, joita C#:ssa nimitetään delegaateiksi. Delegaatti on muuttuja, joka sisältää metodin. Niiden ansiosta metodeita voidaan välittää toisten metodien parametreina, aivan kuin mitä tahansa muitakin muuttujia. (Nagel ym. 2010. 183.) Delegaatit mahdollistavat anonyymien funktioiden tai lambdalausekkeiden käyttämisen, jossa metodeja voidaan välittää parametreina toisille metodeille suoraan**.** (MSDN c)

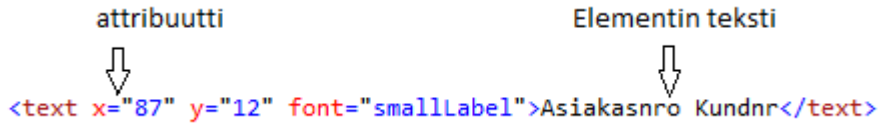
3.2 Visual Studio

Visual Studio on Microsoftin kehittämä integroitu kehitysympäristö (IDE), jolla .NET-sovelluksia kehitetään. Visual Studio sisältää työkalut koodin kirjoittamiseen, graafisen käyttöliittymän luomiseen, lähdekoodin kääntämiseen ja debuggaamiseen sekä automaattisen tekstin täydennyksen. Kalliimmat versiot mahdollistavat myös sovelluksen suorituskyvyn mittaamiseen eli profiloinnin.
Sovelluksesta on saatavilla ilmainen, express-versio, joka tarjoaa perustoiminnallisuudet sovelluksen kehittämiseen. Visual Studiosta on useita maksullisia versioita, jotka tarjoavat hieman toistaan enemmän ominaisuuksia kalliimpaan hintaan (Mackey 2010, 1-2.)

Kalliimmat versiot tarjoavat esimerkiksi työkaluja tiimityöskentelyn helpottamiseen sekä sovelluksen testaamiseen. Ultimate-versio, joka on kaikista kallein mutta sisältää eniten ominaisuuksia, sisältää apuvälineitä myös sovelluksen suorituskyvyn mittaamiseen. (Microsoft.)
Visual Studiota voi muokata ja laajentaa asentamalla lisäosia. Microsoft onkin suunnitellut Visual Studion siten, että sen laajentaminen ja muokkaaminen on helppoa. (Mackey 2010, 9, 25.)

3.3 XML

XML eli Extensible Markup Language on merkintäkieli, jota käytetään tiedon kuvaamiseen ja tallentamiseen sekä tiedon välittämiseen tietojärjestelmien välillä. XML on avoin ja järjestelmäriippumaton teknologia. XML-tiedosto kuvaavat dataa elementtien muodossa. Elementit voiva muodostaa puumaisen rakenteen, jossa elementillä voi olla useita eri alielementtejä, joilla vuorostaan voi olla alielementtejä (WC3 2001.)



Kuva 2. xml-elementin sisällön esittely

XML-elementti voi sisältää sekä yhden tai useampia attribuutteja että elementin tekstin. Kummatkaan niistä eivät kuitenkaan ole pakollisia. (WC3Schools)

XML-spesifikaatio määrittelee vain XML-dokumentin perusrakenteen eli käytännössä elementtien ja attribuuttien oikeaoppisen merkintätavan. Niinpä XML:ää käyttäen on mahdollista kuvata hyvinkin erilaisia ja monipuolisia tietorakenteita. XML:ää onkin käytetty pohjana useille eri tiedon tallennusformaateille sekä tiedonsiirtostandardeille (Cover Pages 2005). Näistä ehkä tunnetuimpia ovat XHTML, RSS ja SOAP. Lisäksi mm. Microsoft Office ja OpenOffice-toimisto-ohjelmistot tarjoavat tiedostojen tallennukseen XML-pohjaista tallennusformaattia. (Cover Pages 2005).

XML-tiedoston rakenteen määrittämiseen voidaan käyttää XML-skeemaa. Skeeman avulla voi kuvata esimerkiksi kaikki elementit ja elementtien attribuutit, joita dokumentissa saa olla. Näin voidaan varmistaa, että XML-dokumentti on oikeassa muodossa eli validia. XML-skeemojen määritykseen on luotu useita eri kieliä. Niistä yleisimmät ovat Document Type Definition eli DTD ja XML Schema Definition eli XSD. (Vlist 2001.)

4 Sovelluksen toteutus

Tämä luku kuvaa sovelluksen suunnittelun, rakenteen sekä toiminnan. Kappaleessa kuvataan, kuinka sovellus hyödyntää xml-tiedostoja, kuvaa käytettävien xml-tiedostojen rakenteen sekä esittelee, kuinka sovellus luo pdf-tiedostoja. Sen lisäksi esitellään sovelluksen toimintaperiaate sekä sovelluksen luokkarakenne. Lopuksi esitellään sovelluksen testaamiseen käytetyt menetelmät.

4.1 XML-asetustiedostojen käyttö

Kuten vaatimusmäärityksessä todettiin, on asiakkaille mahdollista tehdä heille räätälöityjä kolli – eli osoitelappuja. Uusia kollilappuja tullaan luomaan asiakkaille sovelluksen elinaikana mahdollisesti useita, sekä jo olemassa olevia kollilappuja tai rahtikirjapohjia saatetaan muokata. Tämän vuoksi dokumenttien ulkoasu päätettiin kuvata xml-muodossa. Näin dokumenttien ulkoasun kuvaaminen pysyy erillään sovelluksen lähdekoodista, joten lähdekoodia ei tarvitse muokata tai edes kääntää uudestaan, jos dokumenttien ulkoasua halutaan muuttaa. Sovellus ei pidä listaa käytössään olevista asetustiedostoista, vaan tiedosto välitetään sille parametrina, kun sovellusta kutsutaan. Niinpä uuden dokumenttipohjan lisääminen sovellukseen on helppoa. Ainoa vaatimus sovelluksen kannalta on, että asetustiedosto on oikean muotoinen.

Jokaista luotavaa dokumenttipohjaa kohden on yksi xml-tiedosto. XML-tiedostoissa kuvataan dokumentin koko, tyyppi sekä dokumenttiin graafinen sisältö eli piirrettävät viivat, tekstit, mahdolliset kuvat ja viivakoodit. Sovellus lukee XML-tiedostoa rivi kerrallaan ja piirtää rivin sisältämän elementin dokumenttiin.

```
<root doctype="Waybill" pagesize="A4" height="" width="">
  <settings>
    <fonts>
    <font name="smallLabel" type="Verdana" size="5" style="Regular"/>
    <font name="largeLabel" type="Verdana" size="8" style="Bold" />
    <font name="pagenumbering" type="Verdana" size="30" style="Regular" />
    <font name="textcontent" type="Verdana" size="8" style="Regular" />
    </fonts>
    <textholders>
    <textholder>CONSIGNMENT.CONSIGNORCOMPANY</textholder>
    <textholder>CONSIGNMENT.CONSIGNEECITY</textholder>

    <textholder>CONSIGNMENT.DELIVERYCOMPANY</textholder>
    <textholder>CONSIGNMENT.BARCODEID</textholder>
    </textholders>
  </settings>
    <document>
    <line fromX= "18" fromY="32" toX="200" toY="32" lineWidth="1" />
    <line fromX="18" fromY="57" toX="200" toY="57" lineWidth="0,5" />
    <text x="19" y="12" font="smallLabel">Lähettäjä Avsändare</text>
    <text x="19" y="14" font="textcontent">CONSIGNMENT.CONSIGNORCOMPANY</text>
    <barcode type="Code128" x="125" y="62" width="56" height="11">CONSIGNMENT.BARCODEID</barcode>
    <image x="500" y="101" width="30" height="30"  path="C:\PDFManagerLibrary\bin\NB.png" />
```

Kuva 3. Esimerkki rahtikirjan piirtämiseen käytettävästä xml-tiedostosta

XML-tiedoston alussa olevassa root-lohkossa määritetään luotavan dokumentin tyyppi, eli onko muodostettava dokumentti rahtikirja vai osoitelappu, sekä dokumentin koko. Koko voidaan määrittää millimetreinä tai standardeina paperinkokoina, kuten A4 tai A5.

Tämän jälkeen seuraa settings-lohko, jossa kuvataan dokumentin piirtoon käytettävät asetukset, kuten tekstin kirjoitukseen käytettävät fontit sekä varatut sanat. Font-elementti kuvaa fontin nimen, tyypin ja sen koon. Fontteja käytetään tekstin kirjoituksessa siten, että text-elementin font-attribuutin arvoksi annetaan sen fontin nimi, jolla teksti halutaan kirjoittaa.

Textholders-lohkossa määritellään ”varatut sanat”, jotka toimivat linkkeinä lähetyksen tietoihin. Kun lähetykselle luodaan esimerkiksi rahtikirja, text-elemetin kohdalle, jonka tekstinä on esimerkiksi CONSIGNMENT.CONSIGNORCOMPANY, kirjoitetaan lähetyksen lähettävän yrityksen nimi . Nämä varatut sanat siis yhdistetään lähetyksen

tietoihin. Tällä tavoin kirjoitukseen käytettävät fontit sekä muut asetukset ovat aina asetustiedosto – ja dokumenttikohtaisia eikä niitä tarvitse määritellä sovelluksen lähdekoodissa, joten sovelluksen ylläpidettävyys paranee.

Lopulta vuorossa on itse document-lohko, jonka sisällä kuvataan itse dokumentin rakenne ja ulkoasu. Dokument-lohkon sisältämät elementit sisältävät ohjeet itse sisällön piirtämiseksi dokumenttiin. Yksi xml-elementti sisältää käskyn piirtää dokumenttiin yksittäinen teksti, viiva, viivakoodi, kuva jne. Sallitut elementit ovat text, image, barcode, line, pagenumber, goodsline ja goodsline-total. Goodsline- ja goodsline-total-elementtejä tarvitaan rahtikirjan kollirivien piirtämiseen, joten niitä ei käytetä kollilappujen kanssa. Muut elementit sen sijaan ovat yhteisiä sekä rahtikirjoille että kollilapuille. Yhteistä kaikille elementeille on, että ne sisältävät elementin koordinaatit x ja y-akseleilla. Tämän lisäksi eri elementeillä on erilaisia attribuutteja. Esimerkiksi viivan piirtämiseen tarkoitettu line-elementti sisältää tiedon piirrettävän viivan alku- ja loppukoordinaateista, kun taas text-elementti sisältää vain tekstin alkukohdan koordinaatit mutta sisältää toisaalta tiedon käytettävästä fontista.
Text-elementin sisältämä teksti kirjoitetaan dokumenttiin. Tekstinä voi olla myös yllämainitun kaltainen varattu sana, jolloin tekstiksi kirjoitetaan varattua sanaa vastaava lähetyksen tieto.

4.2 PDF-dokumenttien tuottaminen

Pdf-dokumentin luominen ohjelmallisesti on melko haastava tehtävä. Markkinoilla on useita eri ohjelmakirjastoja, jotka yksinkertaistavat pdf-dokumenttien käsittelyä. Yhteistä kaikille näille kirjastoille on, että ne tarjoavat yhtenäisen rajapinnan esimerkiksi grafiikan, tekstin ja kuvien piirtämiseksi pdf-dokumenttiin, sekä keinon tallentaa luotu dokumentti kovalevylle. Tämä helpottaa pdf-tiedostojen luomista huomattavasti, koska tämän jälkeen sovelluskehittäjän tarvitsee vain osata käyttää kirjaston tarjoamaa rajapintaa. Tämän kaltaista valmista ratkaisua käytettiin myös tässä projektissa.

Valitulta kirjastolta edellytettiin, että se tarjoaa suoraviivaisen rajapinnan pdf-dokumentin luomiseen sekä dokumentin sisällön eli tekstin, viivojen sekä myös kuvien

piirtämiseen. Tämän lisäksi edellytettiin, että kirjastolla pystyy tuottamaan tietyntyyppisiä viivakoodeja. Viivakoodeista tärkeimmät olivat GS1-128 sekä Code 39, joita tarvittiin rahtikirjoissa sekä kollilapuissa. Koska pdf-komponentilla tultaisiin luomaan erikokoisia dokumentteja, oli myös tärkeää, että käytettävällä piirtokirjastolla olisi helppo asettaa luotavan dokumentin koko. Dokumentin koko tulisi voida asettaa sekä millimetrin tarkkuudella että standardeina sivukokoina, kuten A4.

Tähän projektiin valittiin ComponentOne PDF for .NET-kirjasto pdf-dokumenttien tuottamista varten. Yksi suurimmista syistä ComponentOne:n tuotteen valintaan oli se, että projektin tilaajalla oli ennestään kokemusta kyseisen yrityksen ohjelmistoista. Tilaajan kokemusten mukaan ComponentOne tarjoaa tuotteilleen myös hyvää asiakastukea. ComponentOne:n pdf-kirjasto ei itse pysty tuottamaan viivakoodeja, mutta ComponentOne:lla on viivakoodien tuottamiseen erillinen kirjasto ComponentOne BarCode for WinForms, jota käytettiin tässä projektissa.

ComponentOne tarjoaa suoraviivaisen rajapinnan pdf-dokumentin luomiseen. Käyttäjä luo aluksi C1PdfDocument-luokan olion, joka käsittää kaiken toiminnallisuuden pdf-tiedoston luomiseen. Luokka sisältää metodit mm. viivojen, tekstin, kuvien tai vaikkapa eri kuvaajien piirtämiseksi dokumenttiin. Kun dokumentti on luotu, se voidaan tallentaa kovalevylle kutsumalla Save-metodia.

4.3 Sovelluksen toiminta ja rakenne

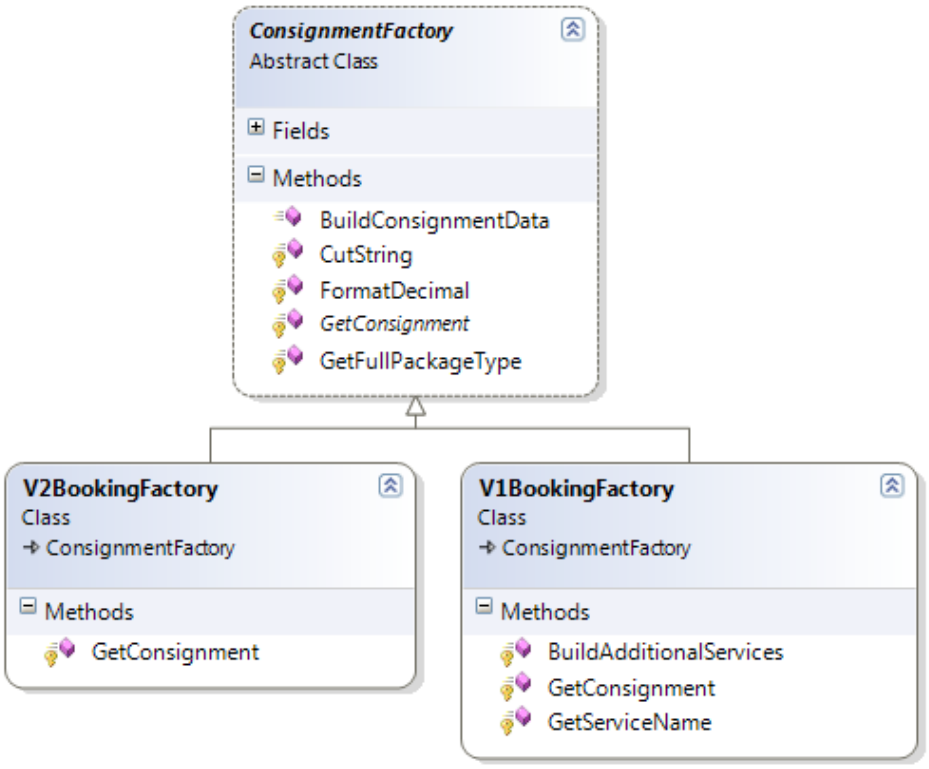
Kuten kappaleessa 2.1 mainittiin, lopullista versiota pdf-komponentista tullaan käyttämään web service -rajapinnan yli. Asiakas pyytää verkkosivun kautta haluamaansa dokumenttia. Verkkosivu kutsuu pdf-komponenttia web service -rajapinnan yli, joka luo pdf-dokumentin ja palauttaa sen verkkosivulle asiakkaan ladattavaksi. Projektin aikana toteutettiin itse pdf-komponentti, web service -rajapinta sekä itse verkkosivusto eivät kuuluneet tämän projektin piiriin.

Pdf-komponenttia kutsuttaessa sille välitetään parametreina lähetyksen tiedot sekä xml-asetustiedosto, jota käytetään dokumentin luomiseen. Tämän jälkeen sovellus käy

asetustiedostoa läpi rivi kerrallaan ja luo asetustiedoston ja lähetyksen tietojen pohjalta pdf-dokumentin. Asetustiedoston käsittely on jaettu kahteen vaiheeseen. Ensimmäisessä vaiheessa käydään läpi tiedoston settings-lohko ja toisessa vaiheessa document-lohko.

Kun dokumentti on luotu, se tallennetaan tietokantaan, jonka jälkeen se annetaan asiakkaalle ladattavaksi. Niinpä jos asiakas haluaa ladata saman dokumentin toistamiseen, sitä ei tarvitse luoda uudestaan, vaan aiemmin luotu dokumentti haetaan tietokannasta. Aina ennen dokumentin luontia sovellus siis tarkistaa tietokannasta, onko kyseiselle lähetykselle jo luotu pyydetyn kaltaista dokumenttia.

4.3.1 Lähetystietojen hakeminen tietokannasta



Kuva 4 Lähetysten hakeminen tietokannasta

Ennen kuin pdf-komponentia voidaan pyytää luomaan pdf-dokumentti, ovat luotavan dokumentin lähetystiedot ensin haettava tietokannasta. Koska sovellus tullaan otta-

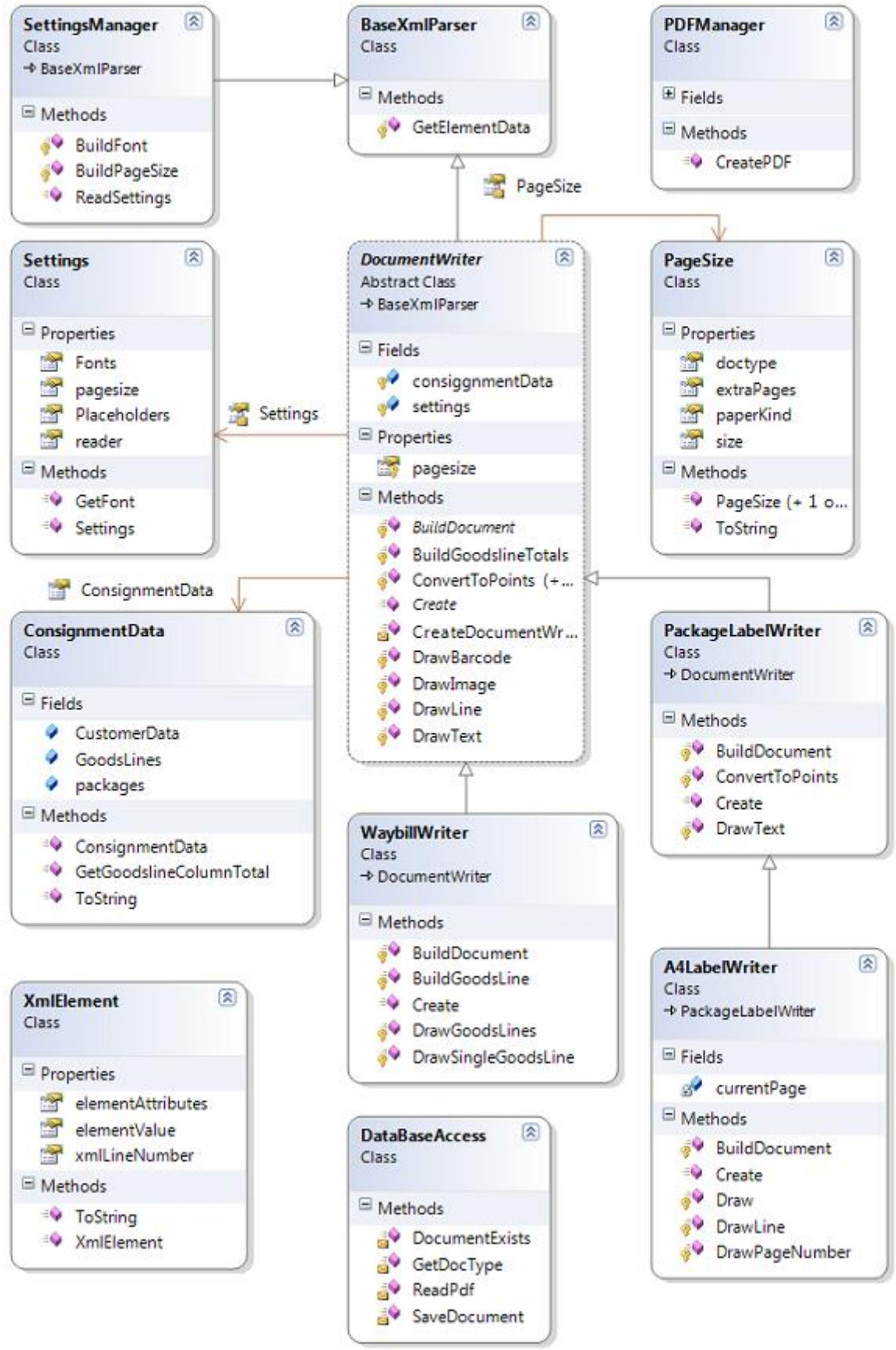
maan käyttöön sekä nykyisessä että kehitteillä olevassa uudessa verkkopalvelussa, tulisi sovelluksen osata lukea lähetyksen tiedot sekä nykyisen verkkopalvelun että rakenteilla olevan uudenverkkopalvelun tietokannasta.

Tämän ongelman ratkaisemiseksi kehitettiin kuvassa 3 esitelty sovellusrakenne. ConsignmentFactory-luokka on abstrakti luokka, joka sisältää metodit lähetyksen datan sisältävän olion luomiseen. Itse datan lukeminen tietokannasta hoidetaan GetConsignment-metodissa, joka on abstrakti metodi, joka aliluokkien on toteutettava. Näin aliluokat voivat hakea dataa eri tietokannasta ja riippumatta tietokannan rakenteesta. Metodi palauttaa lähetyksen tiedot ConsignmentData-luokan muodossa, joka on esitelty luvussa 4.3.3. V1BookingFactory-luokka vastaa lähetystietojen lukemisesta nykyisen verkkopalvelun tietokannasta ja V2BookingFactory vastaa puolestaan tietojen lukemisesta uuden palvelun tietokannasta.

Näiden luokkien toiminta tullaan mahdollisesti myöhemmin siirtämään web service -rajapinnan taakse mahdollisesti omaan luokkakirjastoonsa. Ne eivät siis suoranaisesti ole osa pdf-komponenttia, vaikka pdf-komponentti onkin siitä riippuvainen.

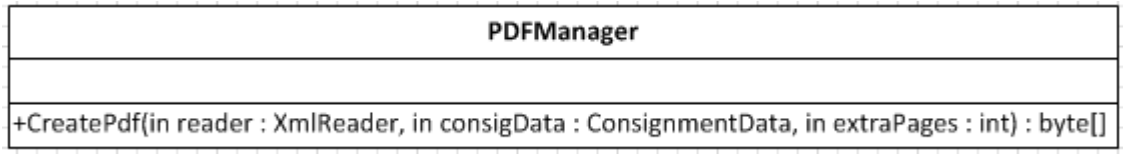
4.3.2 Pdf-komponentin rakenne

Itse sovellus koostuu noin 12 luokasta, jotka on esitelty kuvassa 5. Kuva sisältää sovelluksen kaikki oleellisimmat luokat sekä niiden suhteet toisiinsa. Muutamat vähemmän oleelliset luokat, kuten poikkeuksia esittävät luokat, on jätetty kuvasta pois. Sovelluksen luokkahierarkia on suunniteltu niin, että kaikki luokat, jotka käsittelevät xml-asetustiedostoja periytyvät BaseXmlParser-luokasta. DocumentWriter-luokka on kaikkien pdf-tiedostoja tuottavien luokkien yläluokka. Luokkien välillä on myös muunlaisia suhteita, jotka esitellään seuraavissa kappaleissa.



Kuva 5 Pdf-komponentin luokkakaavio

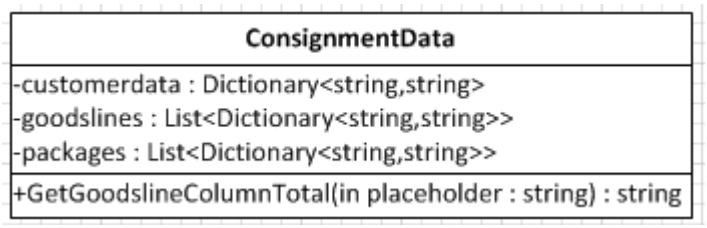
4.3.3 PDFManager



Kuva 6 PDFManager-luokka

PDFManager on sovelluksen ainoa julkinen luokka. Kaikkien muiden sovelluksen luokkien näkyvyysmääre on internal, jolloin ne eivät näy sovelluskirjaston ulkopuolelle. Tämä luokka muodostaa siis sovelluksen koko julkisen rajapinnan. Luokalla on yksi metodi, CreatePDF, jolle sovellusta kutsuttaessa välitetään lähetyksen tiedot sekä dokumentin piirtämiseen käytettävä xml-tiedosto. Mikäli tuotettava dokumentti on rahtikirja, voidaan sovellusta pyytää tuottamaan ylimääräisiä sivuja extraPages-muuttujan avulla. Luokka luo esimerkiksi SettingsManager- ja DocumentWriter-luokien instansseja luodakseen pdf-dokumentin. Kun dokumentti on luotu, se palautetaan kutsujalle tavutaulukkona. Tavutaulukkona on helppo välittää esimerkiksi asiakkaan selaimeen pdf-muodossa.

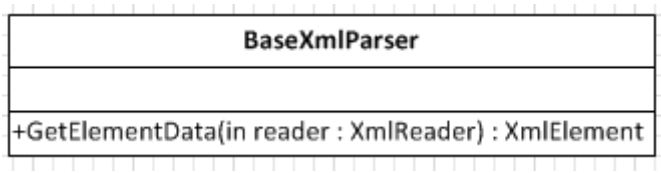
4.3.4 ConsignmentData



Kuva 7 ConsignmentData-luokka

ConsignmentData sisältää lähetyksen kaikki tiedot. Nämä tiedot välitetään pdf-komponentille, kun komponenttia kutsutaan. Lähetyksen tietojen hakeminen tietokannasta kuvattiin kappaleessa 4.3.1 Lähetyksen tiedot sisältävät lähetyksen vastaanottajan, lähettäjän, lähtöpaikan, määräpaikan sekä maksajan tiedot, sekä tiedot esimerkiksi lähetyksen sisällöstä ja lähetyksen mitoista. Lähetyksen tiedot tallennetaan avain-arvopareina.

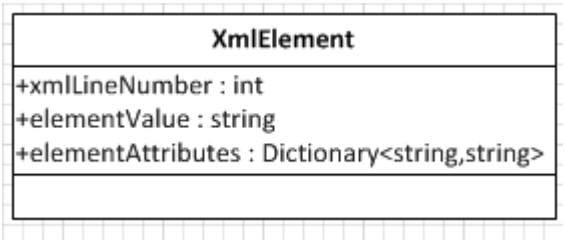
4.3.5 BaseXmlParser



Kuva 8 BaseXmlParser-luokka

Kaikki sovelluksen luokat, jotka käsittelevät xml-tiedostoja, periytyvät BaseXmlParser-luokasta. Luokan ainoa tarkoitus on toimia yliluokkana xml-tiedostoja käsitteleville luokille. Tällä luokalla on vain yksi metodi, GetElementData, jota BaseXmlParser-luokan aliluokat kutsuvat, kun ne haluavat käsitellä xml-tiedostoa. Metodi lukee xml-tiedostosta yhden elementin ja luo sen perusteella XmlElement-olion.

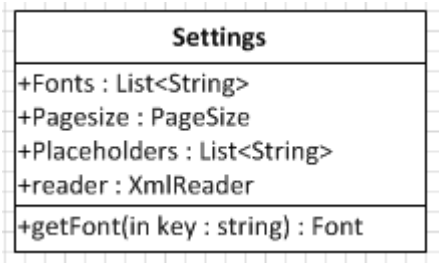
4.3.6 XmlElement



Kuva 9 XmlElement-luokka

XmlElement-luokkaan kääritään yksittäisen xml-elementin sisältämän data. Kuten kappaleessa 4.1 mainittiin, sisältää yksi elementti aina yhden piirto-ohjeen dokumenttiin piirtämistä varten. Xml-elementin sisältämät attribuutit tallennetaan elementAttributes-muuttujaan, johon ne tallennetaan avain-arvo-pareina. Elementin teksti tallennetaan elementValue-muuttujaan. XmlElement-luokan instanssi voidaan välittää DocumentWriter-luokan metodeille, jotka piirtävät pdf-dokumenttiin sen sisällön.

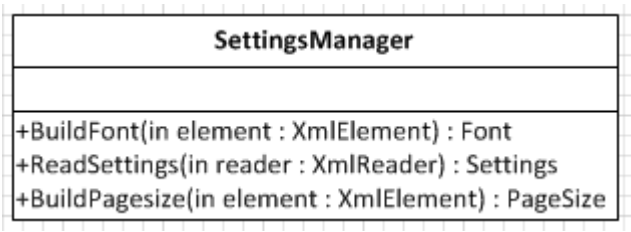
4.3.7 Settings



Kuva 10 Settings-luokka

Settings-luokka sisältää xml-tiedoston settings-lohkon sisältämän datan. Settings-olio sisältää tekstin kirjoitukseen käytettävät fontit sekä tiedon dokumentin koosta sekä siitä, onko kyseessä rahtikirja vai osoitelappu. Reader-muttujaan tallennetaan dokumentin tuottamiseen käytettävä xml-tiedosto. SettingsManager-luokka luo Settings-olion käydessään asetustiedoston settings-lohkoa läpi. Luokan sisältämiä tietoa käytetään pdf-dokumentin luomiseen.

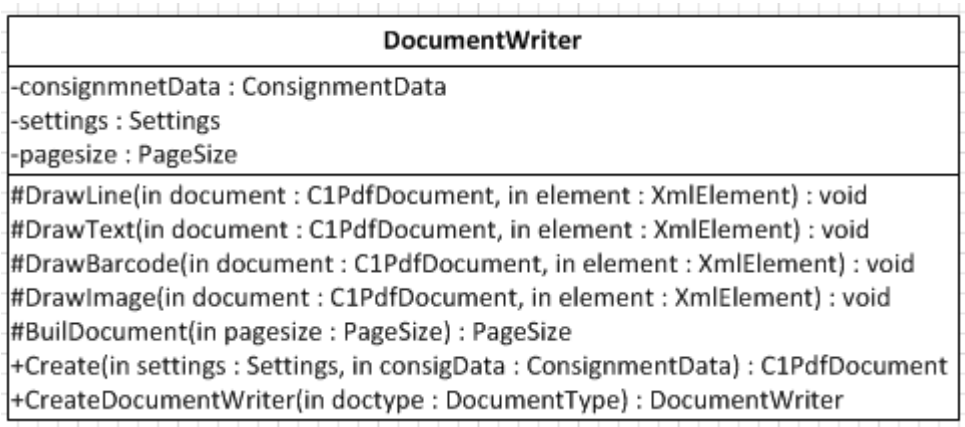
4.3.8 SettingsManager



Kuva 11 SettingsManager-luokka

Xml-asetustiedoston käsittely on jaettu kahteen vaiheeseen. Ensimmäisessä vaiheessa käsitellään tiedoston settings-lohko, ja toisessa vaiheessa käsitellään document-lohko. SettingsManager on vastuussa tästä ensimmäisestä vaiheesta.
Luokan ReadSettings-metodi käy settings-lohkoa läpi rivi kerrallaan lukien tiedon käytettävistä fonteista sekä varatuista sanoista ja luo asetusten perusteella settings-olion.

4.3.9 DocumentWriter



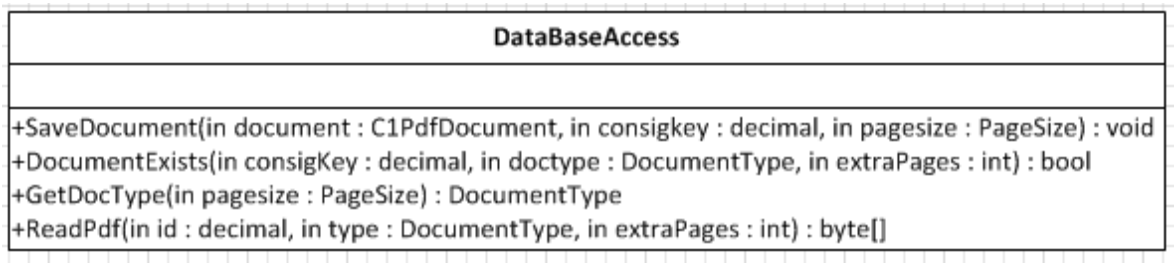
Kuva 12 DocumentWriter-luokka

Kun xml-tiedoston settings-lohko on käsitelty, siirrytään käsittelemään xml-tiedoston document-lohkoa, joka sisältää ohjeet pdf-dokumentin sisällön piirtämiseen. Lohkon käsittelystä vastaa jokin DocumentWriter-luokan aliluokista. DocumentWriter itse on abstrakti luokka, joka sisältää metodit tekstin, viivojen ja muiden kuvioiden piirtämiseksi dokumenttiin, joita luokan aliluokat voivat käyttää. Nämä ns. Draw-metodit kutsuvat ComponentOne:n pdf-kirjaston metodeja pdf-dokumentin sisällön luomiseksi. Luokalla on myös yksi abstrakti metodi, Create, joka luokasta periytyvien aliluokkien on toteutettava.

Luokasta on periytetty kolme aliluokkaa, WaybillWriter, A4LabelWriter ja PackageLabelWriter, jotka vastaavat rahtikirjojen, A4-kokoisten kollilappujen sekä kaikkien muiden kokoisten kollilappujen tuottamisesta. Näitä luokkia ei tässä esitellä sen tarkemmin, sillä niiden toiminta on hyvin lähellä DocumentWriter-luokan toimintaperiaatetta. Kaikki aliluokat toteuttavat DocumentWriter-luokan Create-metodin, joka on abstrakti metodi. Tässä metodissa aliluokat lukevat xml-asetustiedostoa ja kutsuvat yliluokan eri Draw-metodeja. Lisäksi aliluokat voivat ylikirjoittaa yläluokan metodeita. Esimerkiksi kollilappujen piirtämiseen tarkoitettu PackageLabelWriter ylikirjoittaa DrawText-metodin lisäten siihen mahdollisuuden skaalata kirjoitettavaa tekstiä pienemmäksi, mikäli teksti ei muuten mahtuisi sille varattuun kenttään. Tätä toiminnallisuutta ei muita dokumentteja piirtäessä tarvita, joten toiminnallisuus on vain tässä yhdessä aliluokassa

DocumentWriter-luokalla on tehdasmetodi CreateDocumentWriter, joka luo oikean tyyppisen aliluokan DocumentWriter-luokasta. Metodille välitetään parametrina luotavan dokumentin tyyppi, jonka perusteella päätellään, mikä aliluokka luodaan. Näin oikean alikuokan luominen helpottuu sillä luokan kutsujan ei tarvitse itse päätellä, mikä aliluokka olisi oikea

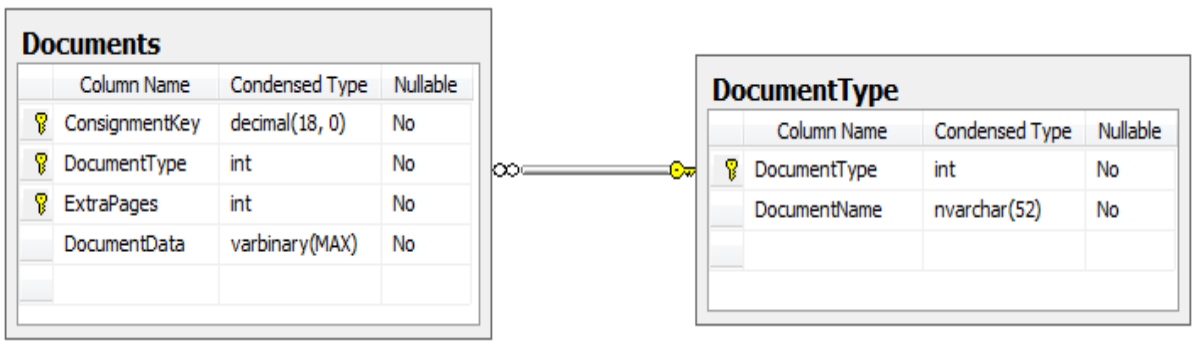
4.3.10 DataBaseAccess



Kuva 13 DataBaseAccess-luokka

DataBaseAccess-luokka on vastuussa valmiin pdf-dokumentin tallentamisesta tietokantaan sekä dokumenttien lukemisesta tietokannasta. Kun dokumentti on luotu, se tallennetaan tietokantaan käyttäen SaveDocument-metodia. Tämän lisäksi aina ennen kuin pdf-dokumentti luodaan, tarkistetaan DocumentExists-metodia käyttäen, löytyykö kyseistä dokumenttia jo tietokannasta. Jos dokumentti löytyy, sitä ei tarvitse luoda uudestaan vaan dokumentti voidaan hakea käyttäjälle suoraan tietokannata käyttäen ReadPdf-metodia.

4.4 Tietokanta



Kuva 14 Tietokannan relaatiokaavio

Sovelluksen luomat pdf-tiedostot tallennetaan tietokantaan. Sovelluksen lopullisessa versiossa dokumentit tallennetaan SQL Server 2012-tietokantaan, mutta sovelluksen kehityksen aikana tiedostot tallennettiin SQL Server 2008 R2-tietokantaan, koska uudempi tietokantapalvelin ei sovelluksen kehityksen aikana ollut vielä käytettävissä. Sovelluksen tietokanta muodostuu kahdesta taulusta. Documents-taulu sisältää itse pdf-tiedostot, jotka tallennetaan tietokantaan binäärimuodossa. DocumentType-sarake kertoo tallennetun dokumentin tyypin eli onko dokumentti rahtikirja tai esimerkiksi A5-kokoinen kollilappu. DocumentType-sarakkeen ja DocumentType-taulun välillä on relaatio. Documents-taulun DocumentType on viiteaavain, joka viittaa DocumentType-tauluun. Näin DocumentType-kentässä ei voi olla muita arvoja kuin mitä DocumentType-taulussa on määritelty.

4.5 Sovelluksen testaaminen

Sovellusta testattiin kehityksen aikana tiiviisti. Ensinnäkin pdf-komponentin kaikki metodit ajettiin debuggerin läpi useaan kertaan eri parametreilla, jotta varmistuttiin, että metodit toimivat kuten oli odotettu.

Testaamiseen ei käytetty varsinaisia yksikkö –tai järjestelmätestejä, mutta sovelluksen testaamista varten kirjoitettiin yksinkertainen sovellus, joka hakee tietokannasta sattumanvaraisia lähetyksiä ja luo niiden pohjalta sovellusta käyttäen lähetyksen rahtikirjan sekä A5-kokoisen osoitelapun. Näitä dokumentteja verrattiin sillä hetkellä tuotantokäytössä olleen pdf-komponentin luomiin dokumentteihin ja varmistettiin, että ne vastasivat toisiaan. Koska testaamisessa käytetyt lähetykset valittiin sattumanvaraisesti, löydettiin sovelluksesta testauksen aikana useita virheitä, jotka johtuivat siitä, ettei sovellus ottanut huomioon kaikkia harvinaisempia poikkeustapauksia.

Samalla testisovelluksella testattiin myös sovelluksen suorituskykyä tuottamalla useita dokumentteja peräkkäin ja mittaamalla yhden dokumentin tuottamiseen kulunut keskimääräinen aika. Näiden testien perusteella arvioitiin, että uusi pdf-komponentti on jonkin verran vanhaa komponenttia nopeampi. Vertailukelpoisia aikoja ei kuitenkaan saatu mitattua, joten tarkempaa nopeusvertailua ei voitu suorittaa.

5 Pohdinta

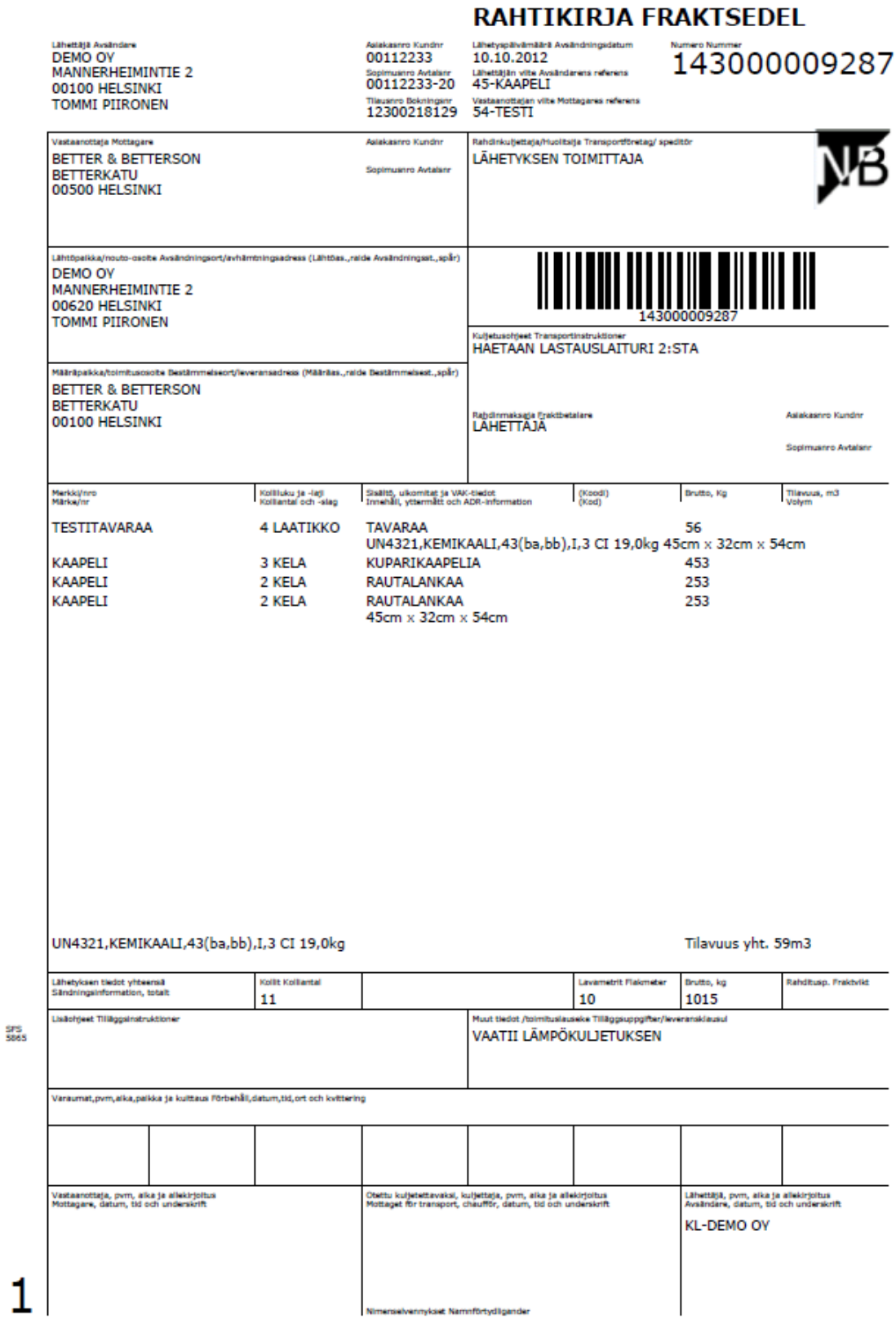
Projektin tarkoituksena oli luoda asiakkaalle ohjelmistokomponentti pdf-dokumenttien tuottamista varten. Projekti kattoi sovelluksen suunnittelun, kehittämisen ja testaamisen. Sovelluksen käyttöönotto ja integrointi muihin tietojärjestelmiin rajattiin tämän projektin ulkopuolelle. Projekti aloitettiin vuoden 2012 syksyllä ja suunniteltu kesto projektille oli n. 5 kuukautta eli projektin suunniteltu valmistumisajankohta oli vuoden 2013 alkupuolella.

5.1 Projektin tulokset

Projektin aikana toteutettiin sovellus, jolla voi tulostaa rahtiliikenteessä käytettäviä rahtikirjoja sekä kolli- eli osoitelappuja. Tilaajan toiveiden mukaisesti kehitystyön aikana keskityttiin A4-kokoisten rahtikirjojen sekä A5-kokoisten kollilappujen tuottamiseen. Näille dokumenteille luotiin myös xml-asetustiedostot. Luotu sovellus pystyy lukemaan lähetystietoja sekä tämän projektin aikana käytössä olleesta tietokannasta, sekä uudesta vielä työn alla olevasta uuden verkkopalvelun tietokannasta. Dokumenttien tuottamista testattiin myös perinpohjaisesti, joten näiden dokumenttien osalta sovellus on tuotantokelpoinen.

Sovelluksella voi tuottaa myös muunkokoisia kollilappuja, mutta niille ei projektin aikana luotu lopullisia asetustiedostoja. Projektin aikana luodut asetustiedostot ovat vain testikäyttöön tarkoitettuja ja ovat siten keskeneräisiä. Tämän lisäksi erikokoisten dokumenttien tuottamista ei aikataulullisista syistä testattu kovinkaan tarkasti. Lisäksi näiden dokumenttien tuottaminen oli tilaajalle vasta toissijainen tavoite, joten on mahdollista, että näiden dokumenttien luominen ei vielä toimi kaikilta osin toivotulla tavalla.

Sovelluksella pystyy tuottamaan myös asiakkaille erikseen muotoiltuja kollilappuja, mutta asetustiedostoja ei näille dokumenteille ole vielä tehty yksinkertaisia testidokumentteja lukuun ottamatta.



RAHTIKIRJA FRAKTSEDEL

Lähettäjä Avsändare
DEMO OY
MANNERHEIMINTIE 2
00100 HELSINKI
TOMMI PIIRONEN

Asiakasnro Kundnr
00112233
Sopimusnro Avtalsnr
00112233-20
Tilausnro Bokningsnr
12300218129

Lähetyspäivämäärä Avsändningsdatum
10.10.2012
Lähettäjän viite Avsändarens referens
45-KAAPELI
Vastaanottajan viite Mottagarens referens
54-TESTI

Numero Nummer
143000009287

Vastaanottaja Mottagare
BETTER & BETTERSON
BETTERKATU
00500 HELSINKI

Asiakasnro Kundnr
Sopimusnro Avtalsnr

Rahdinkuljettaja/Huoltaja Transportföretag/ speditör
LÄHETYKSEN TOIMITTAJA

Lähtöpaikka/nouto-osoite Avsändningsort/avhämtningsadress (Lähtöas.,raide Avsändningsst.,spår)
DEMO OY
MANNERHEIMINTIE 2
00620 HELSINKI
TOMMI PIIRONEN

143000009287

Kuljetusohjeet Transportinstruktioner
HAETAAN LASTAUSLAITURI 2:STA

Määräpaikka/toimitusosoite Bestämmelseort/leveransadress (Määräas.,raide Bestämmelsest.,spår)
BETTER & BETTERSON
BETTERKATU
00100 HELSINKI

Rahdinmaksaja Fraktbetalare
LÄHETTÄJÄ

Asiakasnro Kundnr
Sopimusnro Avtalsnr

Merkki/nro Märke/nr	Kolliluku ja -laji Kolliantal och -slag	Sisältö, ulkomitat ja VAK-tiedot Innehåll, yttermått och ADR-information	(Koodi) (Kod)	Brutto, Kg	Tilavuus, m3 Volym
TESTITAVARAA	4 LAATIKKO	TAVARAA UN4321,KEMIKAALI,43(ba,bb),I,3 CI 19,0kg 45cm x 32cm x 54cm		56	
KAAPELI	3 KELA	KUPARIKAAPELIA		453	
KAAPELI	2 KELA	RAUTALANKAA		253	
KAAPELI	2 KELA	RAUTALANKAA 45cm x 32cm x 54cm		253	

UN4321,KEMIKAALI,43(ba,bb),I,3 CI 19,0kg

Tilavuus yht. 59m3

Lähetyksen tiedot yhteensä Sändningsinformation, totalt	Kollit Kolliantal		Lavametrit Flakmeter	Brutto, kg	Rahditusp. Fraktvikt
	11		10	1015	

SFS 5865

Lisäohjeet Tilläggsinstruktioner

Muut tiedot /toimituslauseke Tilläggsuppgifter/leveransklausul
VAATII LÄMPÖKULJETUKSEN

Varaumat,pvm,aika,paikka ja kuittaus Förbehåll,datum,tid,ort och kvittering

Vastaanottaja, pvm, aika ja allekirjoitus
Mottagare, datum, tid och underskrift

Otettu kuljetettavaksi, kuljettaja, pvm, aika ja allekirjoitus
Mottaget för transport, chaufför, datum, tid och underskrift

Nimenselvennykset Namnförtydliganden

Lähettäjä, pvm, aika ja allekirjoitus
Avsändare, datum, tid och underskrift
KL-DEMO OY

1

Kuva 15 Esimerkki sovelluksen tuottamasta SFS 5865-standardiin perustuvasta rahtikirjasta

5.2 Sovelluksen jatkokehitysehdotuksia

Vaikka projektin aikana valmistunut sovellus toimiikin suunnitellusti, ei sovellus vielä kuitenkaan ole täysin valmis. Tällä hetkellä sovellus pystyy tuottamaan vain yhdentyyppisiä kollilappuja, koska tarvittavia xml-asetustiedostoja ei vielä ole tehty muille kollilapputyypeille. Lisäksi esimerkiksi A4-kokoisten kollilappujen tuottamista ei ole testattu kunnolla, joten on mahdollista että sovellusta joudutaan vielä parantelemaan, jotta A4-kollilaput voitaisiin tuottaa ongelmitta.

Nyt valmistunut sovellus on osa tilaajan verkkopalveluiden uudistamissuunnitelmaa. Jotta sovelluksesta olisi tilaajalle hyötyä, tulisi se integroida osaksi verkkopalveluita. Tämä tarkoittaa esimeriksi web-service- rajapinnan ohjelmointia pdf-komponentin ja verkkopalveluiden väliin, tai sovelluksen käyttämistä muulla tavoin. Sovellus tullaan ottamaan käyttöön projektin valmistuttua.

5.3 Projektin aikana opittua

Opinnäytetyöprojekti on ollut pitkä ja opettavainen. Projektin aikana opin paljon projektityöskentelystä sekä ohjelmistokehityksestä.
Vaikka koulukursseilla olikin opiskeltu projektityöskentelyä ja projektin hallintaa, olivat ne ehkä jääneet itselläni koulussa vähemmälle huomiolle. Niinpä tämän projektin läpivienti opetti minulle projektityöskentelystä paljon. Opin projektin suunnittelusta ja projektin hallinnasta ja seurannasta paljon uutta. Koska projekti kesti n. 5 kuukautta, eli n. 400 työtuntia, myös ajankäyttö aiheutti yllätyksiä. Välillä alkoi vaikuttaa jopa siltä, ettei projekti valmistuisi ajallaan, mutta paremmalla ajankäytön suunnittelulla vaikeudet voitettiin.

Sovelluksen suunnittelu ja toteutus olivat kaikkein mielenkiintoisimmat vaiheet projektissa. Sovelluksen rakenteesta täytyi tehdä tarpeeksi joustava, jotta samalla ohjelmalla pystyisi tuottamaan kaikki vaadittavat erityyppiset dokumentit. Lisäksi dokumenttien ulkoasun määrittely ja muokkaaminen tuli olla mahdollisimman suoraviivaista ja yksinkertaista, mutta toisaalta xml-tiedostossa tuli voida ilmaista kaikki tarvittavat pdf-

dokumentin rakenteet, jotta dokumenttien ulkoasun määrittely pysyisi erillään sovelluksen lähdekoodista. Lisäksi lähetystietojen hakemisen täytyi onnistua sekä nykyisestä että kehitteillä olevasta tietokannasta.

Pdf-dokumentteja ja xml-tiedostoa käsittelevien luokkien muodostaman hierarkian suunnittelu ja toteutus oli hyvin opettavaista. Vaikka olin ennen tätä projektia ohjelmoinut melko paljon ja nimenomaan olio-ohjelmointikielillä, en aiemmin ollut rakentanut näin monimutkaista luokkahierarkiaa. Mielestäni rakenne oli toimiva. Onnistuin mielestäni melko hyvin abstrahoimaan matalamman tason toiminnallisuudet yläluokkiin ja käyttämään niitä pohjana monimutkaisempien toiminnallisuuksien toteuttamiseksi olio-ohjelmoinnin periaatteiden mukaisesti.

Teknisen toteutuksen kanssa ilmeni myös yllätyksiä. Suurin ongelma projektin kannalta oli, etten sovellusta suunnitellessani ottanut huomioon läheskään kaikkia erilaisista lähetyksen tiedoista johtuvia erikoistapauksia, joita rahtikirjaa tai kollilappua luodessa voi tulla vastaan. Esimerkiksi jos lähettäjän osoitetiedot ovat liian pitkät, tulee ne pilkkoa useammalle riville, jotta ne mahtuisivat osoitetiedoille varattuun kenttään. Tämän tyyppisiä tilanteita tuli vastaan useita, ja ongelmat huomattiin usein vasta sovelluksen testaamisen aikana, koska ne saattoivat ilmetä vain suhteellisen harvojen lähetysten kohdalla. Tämän seurauksena jo valmiiksi luultuja sovelluksen osia jouduttiin muokkaamaan pahimmillaan hyvinkin paljon, joka aiheutti lisätyötä ja muutoksia sovelluksen rakenteeseen. Mikäli teknistä toteutusta suunnitellessa näitä poikkeustapauksia olisi kartoitettu tarkemmin, olisi säästytty paljolta ylimääräiseltä työltä.

Lähteet

Cover Pages 2005. XML Applications and Initiatives. Luettavissa: http://xml.coverpages.org/xmlApplications.html. Luettu: 14.1.2013.

ECMA International 2012. Standard Ecma-335 Common Language Infrastructure (CLI). Luettavissa http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf Luettu 8.1.2013.

IronPython 2012. Luettavissa: http://ironpython.net/ Luettu: 7.1.2013.

Liberty, J. 2001. What You Need to Know to Move from C++to C#. MSDN Magazine. Luettavissa: http://msdn.microsoft.com/en-us/magazine/cc301520.aspx Luettu: 4.1.2013.

Mackey, A. 2010. Introducing .NET 4.0 With Visual Studio 2010. Apress. USA.

Microsoft. Visual Studio Compare Editions. Luettavissa: http://www.microsoft.com/visualstudio/eng/products/compare Luettu: 8.1.2013.

MSDN 2003. An Overview of Managed/Unmanaged Code Interoperability Luettavissa: http://msdn.microsoft.com/en-us/library/ms973872.aspx Luettu: 4.1.2013.

MSDN 2006. Introducing Windows Presentation Foundation. Luettavissa: http://msdn.microsoft.com/en-us/library/aa663364.aspx Luettu: 4.1.2013.

MSDN a. Common Language Runtime (CLR). Luettavissa: http://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx Luettu: 2.1.2013.

MSDN b .NET Framework Versions and Dependencies, Luettavissa: http://msdn.microsoft.com/en-us/library/bb822049.aspx Luettu: 3.1.2013.

MSDN c .Lambda Expressions (C# Programming Guide). Luettavissa: http://msdn.microsoft.com/en-us/library/bb397687.aspx. Luettu: 9.1.2013

Mono 2012a. What is Mono. Luettavissa: http://www.mono-project.com/What_is_Mono Luettu: 3.1.2013.

Mono 2012b. Compatibility. Luettavissa: http://www.mono-project.com/Compatibility. Luettu: 8.1.2013.

Nagel, C., Evjen, B., Glynn, J., Watson, K., Skinner, M. 2010. Professional C# 4 and .NET 4. Wiley Publishing, Inc. USA.

Vlist, E. 2001. Comparing XML Schema Languages. O'Reilly Media Inc. USA. Luettavissa: http://www.xml.com/pub/a/2001/12/12/schemacompare.html. Luettu: 14.1.2013.

WC3Schools, XML Elements, Luettavissa: http://www.w3schools.com/xml/xml_elements.asp. Luettu: 9.1.2013.

WC3 2001. XML in 10 points. World Wide Web Consortium (W3C). Luettavissa: http://www.w3.org/XML/1999/XML-in-10-points.html.en. Luettu: 9.1.2013.